2.5D Clip-Surfaces for Technical Visualization

Matthias Trapp Jürgen Döllner
Hasso-Plattner-Institut, University of Potsdam, Germany
{matthias.trapp | juergen.doellner}@hpi.uni-potsdam.de

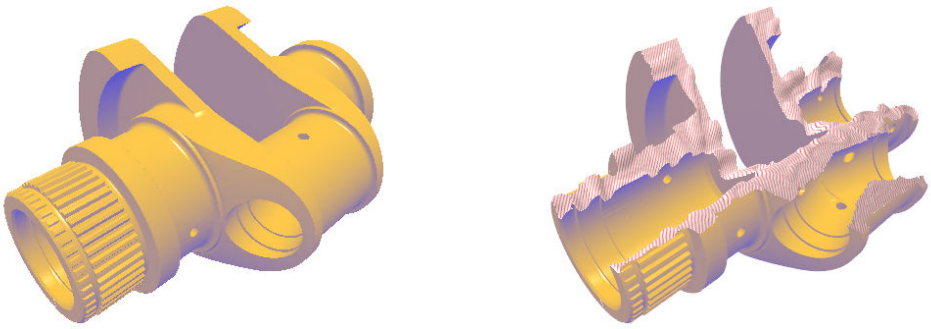


Figure 1: Application of a 2.5D clip-surface to a virtual 3D model of a crank.

ABSTRACT

The concept of clipping planes is well known in computer graphics and can be used to create cut-away views. But clipping against just analytical defined planes is not always suitable for communicating every aspect of such visualization. For example, in hand-drawn technical illustrations, artists tend to communicate the difference between a cut and a model feature by using non-regular, sketchy cut lines instead of straight ones. To enable this functionality in computer graphics, this paper presents a technique for applying 2.5D clip-surfaces in real-time. Therefore, the clip plane equation is extended with an additional offset map, which can be represented by a texture map that contains height values. Clipping is then performed by varying the clip plane equation with respect to such an offset map. Further, a capping technique is proposed that enables the rendering of caps onto the clipped area to convey the impression of solid material. It avoids a re-meshing of a solid polygonal mesh after clipping is performed. Our approach is pixel precise, applicable in real-time, and takes fully advantage of graphics accelerators.

Keywords: clipping planes, real-time rendering, technical 3D visualization

1 INTRODUCTION

Clipping planes are often used in illustrative technical visualization for the exploration of complex 3D shapes. Resolving spatial occlusion, they enable a user to view the internal or hidden parts and assembly processes of computer-aided design drawings more efficiently by identifying any intersection that conflicts within the assembly or the assembly configurations. This clipping mechanism is a fast and easily understandable technique with simple interaction [2]. There are a number of clipping variants, such as clipping into half-spaces, cross sections, exploded or cutaway-views, which have proven successful in various areas of visualization.

However, traditional planar clipping planes might not always be suitable to communicate every aspect of an illustrative technical visualization. For example, in hand-drawn technical illustrations, artists tend to communicate the difference between a cut and a model feature by using non-regular, sketchy or ragged cut lines instead of straight ones [23]. To enable the rendering of such effect, this papers presents an extension of the planar clipping planes that is denoted as *2.5D clip-surface* (CS) (Figure 1). It enables clipping against a number of non-planar, more precisely 2.5D clipping planes, and the flexible stylization of the cutting regions. The presented approach is suitable to extend existing rendering techniques for interactive cut-away views or cross-section illustrations.

The basic idea of a 2.5D CS is simple: consider the plane equation for a traditional clipping plane $CP = (N_x, N_y, N_z, D)$. Using this parametric plane equation, one can decide for every given position $P = (x, y, z) \in \mathbb{R}^3$ in which half space it is located. Clipping can then be formulated as follows:

$$\chi(CP, P) = N_x \cdot x + N_y \cdot y + N_z \cdot z + D > 0$$

This formulation can be extended by varying the distance D to the plane origin for each point P by using an additional height value $h = f(CP, P, OM)$ that is sampled from a offset texture map (OM) using f. This results in a new parameterization:

$$\chi(CS, P) = N_x \cdot x + N_y \cdot y + N_z \cdot z + D + h > 0$$

By introducing local distance variations, 2.5D clip-surfaces can be used to interactively create non-regular

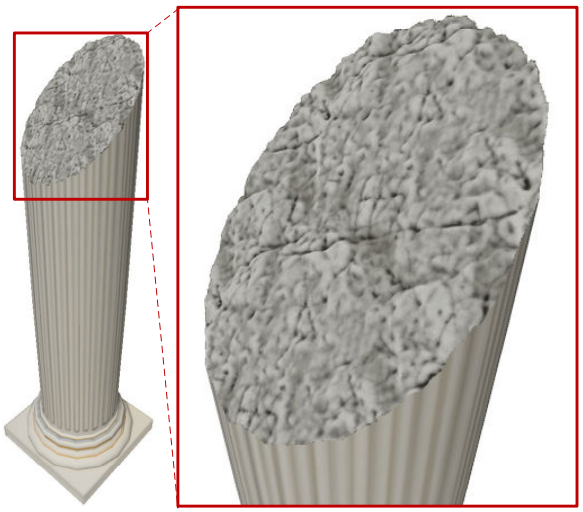


Figure 2: Application of a 2.5D clip-surface and with a cap surface to a column dataset.

clipping silhouettes while a cap-surface conveys complex inner structure of a clipped mesh (Figure 2).

While the application of CS can be implemented straight forward using fragment shader programs, the task of conveying the impression of solid material by using a *cap surface* to visualize the inner structure of an input mesh, is a challenging one. Due to the non-planarity of the clip surface, capping techniques based on stencil-buffer capabilities [3] cannot be applied to non-convex shapes, because the association of a cap surface to a clipped area cannot be decided in image-space using stencil masks.

In this work we present a concept and rendering technique that addresses the above challenges with respect to real-time, raster-based image synthesis. To summarize, this work makes the following contributions:

1. It presents a fully-hardware accelerated rendering technique that enables the application of multiple 2.5D clip-surfaces within a single rendering pass.

2. It presents a novel concept and fully hardware accelerated implementation for generating and rendering of cap surfaces.

The remainder of this paper is structured as follows: Section 2 discusses related and previous work. Section 3 introduces a rendering pipeline that implements 2.5D clip-surfaces with cap surfaces. Section 4 describes the parametrization and hardware-accelerated real-time implementation of clip-surfaces. Section 5 describes the generation and rendering of cap-surfaces. Section 6 presents application examples, discusses problems and limitations, and presents ideas for future work. Finally, Section 7 concludes this paper.

2 RELATED WORK

Despite the fundamentals of line and polygon clipping [9, 11] for polygonal 3D scene representations, related work mostly comprises the generation of *cut-away views* and *exploded view diagrams*. These interactive visualization techniques reveal the interior of complex 3D models by clipping either occluding parts or outer layers [19]. Traditionally, these depictions are static and often hand-crafted, thus the view point and the displayed cuts are fixed. Interactive cut-away views overcome these drawbacks by enabling the user to choose desired cut planes and cut volumes, while exploring and navigating the 3D virtual environment simultaneously.

In 1993, Lorensen presents an approach for image-based clipping using Boolean textures [20], a texture mapping technique, which is based on implicit functions to generate texture maps that are used to perform clipping during a renderer's scan conversion step. However, implicit functions are usually hard to model. Coffin et al. present a technique that enables a user to look beyond occluding objects in arbitrary 3D graphics scenes by interactively cutting holes into the occluding geometry [7]. The interactive image synthesis for this kind of virtual x-ray vision is performed using constructive solid geometry (CSG). Also using image-based CSG for rendering [14], a sophisticated approach for generating 3D cut-away views was introduced by Li et al. [19]. This work was later extended to generate interactive explosion diagrams [18]. An application of this rendering technique to mathematical surfaces [13] uses planar clipping planes. All of the previous approaches do not directly enable the stylization of the cut surfaces and the rendering performance depends on the depth complexity of the virtual scene with respect to the virtual camera [14].

A more effective image-based approach for rendering cut-aways views for geometrical complex 3D scenes is presented by Burns et al. [5]. Based on distance transformations of the depth buffer content, view-dependent cut-aways can be generated for a number of 3D objects. This approach depends on the virtual camera and always exposes the complete objects-of-interest in the context of surrounding objects. Our approach enables the creation of consistent cut-away illustration for varying virtual cameras. In [15], a system for creating illustrative cutaway renderings is presented that rely on sketch-based interfaces and stylized rendering techniques for the study of elaborate 3D models. With respect to interaction, Clifton & Pang present extensions to the traditional cutting plane for virtual reality devices. Using their hands, users interact directly with the data to generate arbitrarily oriented planar surfaces, linear surfaces, and curved surfaces [6].

In the field of volume graphics and rendering, Weiskopf et al. propose clipping methods that are

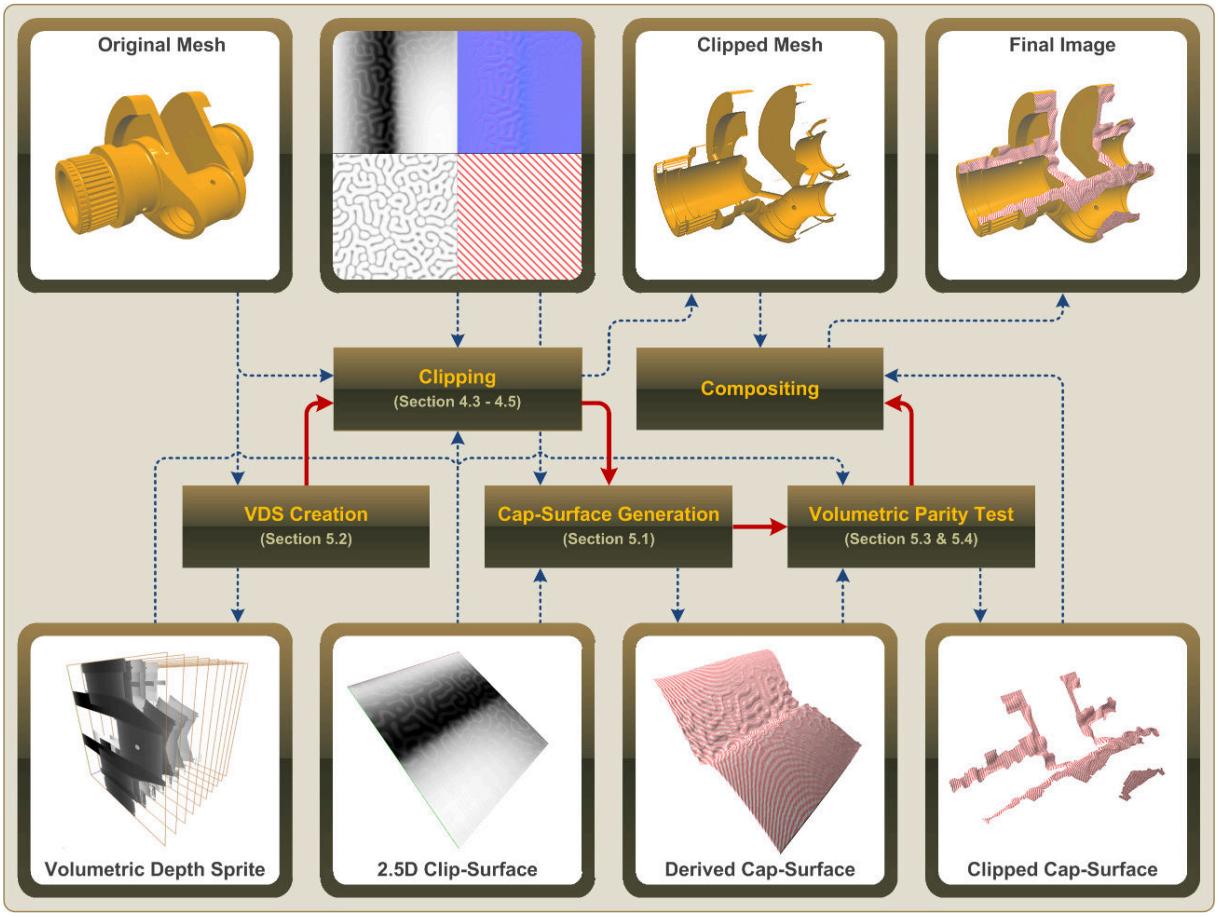


Figure 3: Conceptual overview of the rendering pipeline for 2.5D clip-surfaces and non-planar cap surfaces.

capable of using complex geometries for depth-based volume clipping [32]. It is based on an approach for volume clipping based on per-fragment operations on voxelized clip object that are used to identify the clipped regions [31]. Based on over-sampling fragment shader programs, Röttger et al. increase the rendering quality for volumetric clipping [27]. Further, Qi & Martens investigate the aspects of positioning a clipping plane within volume-rendered data. They propose three different interface prototypes that combine aspects of 2D graphical user interfaces with 3D tangible interaction devices based on wireless vision-based tracking [26]. Birkeland et al. presents a feature and context preserving clipping approach called membrane clipping [2]. These non-planar clipping planes implement selective feature preservation using an elastic membrane. With respect to the cut-away approaches described above, our contribution presents a complementing approach for creating cut-away illustration.

3 CONCEPTUAL OVERVIEW

The proposed image-based approach works for an arbitrary solid input mesh. Figure 3 shows the rendering pipeline for 2.5D clip-surfaces and cap surfaces applied to the domain of technical illustration. The complete rendering process comprises the following three steps that are performed on a per-frame basis:

1. Application of clip surfaces by rendering the solid mesh into the framebuffer with applied clipping (Section 4).

2. Automatic generation of a cap surface from the clip-surface. This is implemented using a polygonal cap surface that is derived from the clip-surface parametrization. GPU based-mesh refinement [4] is applied to fit the subdivision of the cap mesh to the resolution of the offset map (Section 5.1).

3. Clipping of the cap mesh by performing a *volumetric depth test* on a per-fragment basis. It determines if a fragment lies inside the volume and thus is associated with a gap, or if it is located outside the input shape and therefore is discarded. In this step per-vertex displacement mapping, and per-fragment shading, and texturing is also performed (Section 5.2 and 5.3).

Subsequently, the intermediate results of each stage (i.e., the fragments of the clipped mesh and clipped cap-surface) can be composited by rendering to the frame

buffer successively or by using an additional compositing step to apply post-processing effects (e.g., edge-enhancement [22]). Therefore, the intermediate results are rendered into off-screen buffers [28] and composited using an additional post-processing pass.

4 IRREGULAR CLIP-SURFACES

As described previously, a 2.5D clip-surface is an extension of the standard clipping plane by offsetting each point on the plane using height variances. Instead of modeling these local height variances using implicit functions [20] they are represented using a texture map that contains height values.

Given a surface parametrization (Section 4.1) including a respective offset map (Section 4.2), pixel-precise clipping (Section 4.3) can be performed for every fragment position during rasterization using a fragment shader program (Section 4.4). This approach also enables the rendering of multiple clip-surfaces within a single rendering pass (Section 4.5).

4.1 Parameterization

Briefly, a 2.5D clip-surface $CS = (O,U,V,S,OM)$ can be modeled using the following five parameters (cf. Fig. 4): an origin $O \in \mathbb{R}^3$, two orthogonal direction vectors $U,V \in \mathbb{R}^3$ (to support anisotropic adjustments), a scaling vector $S = (S_x,S_y,S_z) \in \mathbb{R}^3$, and an offset texture map OM. The normal vector N of the plane is implicitly given by $N = \|U\| \times \|V\|$. A CS can be extended with further attributes (e.g., normal, diffuse, and light texture maps) that define the appearance of the cap surface.

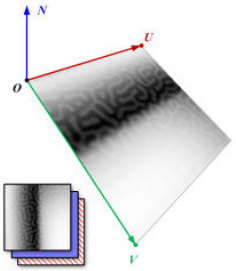


Figure 4: Parameters of a 2.5D clip-surface.

4.2 Offset Maps

The top row of the right Figure 5 shows some examples of 2D offset maps. Basically, it is an image-based representation of a 2.5D *clip surface* that is used for clipping. Offset maps can be easily combined (e.g., Column 3), using standard image blending operations. The accompanying normal maps (Row 2) and occlusion maps (Row 3) can be computed based on the offset maps using a preprocessing step at run-time.

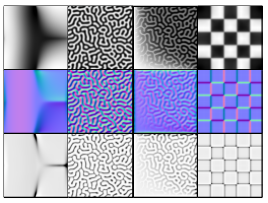


Figure 5: Examples of different offset, normal, and occlusion maps (top to bottom).

Representing the irregular clip surface using texture maps offers a number of advantages: (1) it enables a high design freedom since they can be created using standard imaging software; (2) they can be easily exchanged and combined; and (3) can be easily represented and modified on GPU.

4.3 Pixel-precise Clipping

Given an arbitrary shaped solid mesh and a CS, clipping is performed at fragment level. For each fragment with a clip-space coordinate P the following Boolean function:

$$\chi(CS,P) = \begin{cases} 1 & P \bullet N - N \bullet O + f(OM,T) \cdot S_z < 0 \\ 0 & otherwise \end{cases} \quad (1)$$

$$T = \left(\frac{(P_N - O) \bullet \hat{U}}{\|U\|} \cdot S_x, \frac{P_N - O \bullet \hat{V}}{\|V\|} \cdot S_y \right) \quad (2)$$

$$P_N = P - ((P - O) \bullet N) \cdot N \quad (3)$$

is evaluated. Here, the function f delivers a scalar value $h \in [0,1]$ by first, generating texture coordinates T using Equation 2 and 3, and second, sampling the offset map OM, and finally scaling the resulting height sample by S_z. The sampling step depends on the type of offset map used, i.e., it differs for 1D, 2D, or 3D representations for an offset map. A point P is clipped if the half-space test in Equation 1 resolves to 1 (*true*). This function can be efficiently implemented using a fragment shader program and the fragment discard functionality.

4.4 Fragment Shader Implementation

Listing 1 shows an OpenGL shading language (GLSL) implementation for the clip-surface parameterization. It first assembles the plane equation from the parameterization, then project the input point onto that plane and compute the required texture coordinates. Using these, the offset texture map is sampled and half-space test is performed. This function can be executed for a number of surfaces within a single rendering pass. Prior to shader execution, the parameterization is encoded in a matrix representation.

```
bool clipSurface(
 const in mat4 config,
 const in vec4 P,
 const in sampler2D offsetMap){
 // compute plane parametrization in eye space...
 vec3 O= (gl_ModelViewMatrix *
          vec4(config[0].xyz, 1.0)).xyz;
 vec3 A= gl_NormalMatrix*normalize(config[1].xyz);
 vec3 B= gl_NormalMatrix*normalize(config[2].xyz);
 vec3 N= cross(A, B);
 // project fragment coordinates onto plane
 vec3 PN= P.xyz-dot(P.xyz-O, N)*N;
 // compute clip texture coordinates...
 float s= dot(PN-O, A) / length(config[1].xyz);
 float t= dot(PN-O, B) / length(config[2].xyz);
 // fetch height...
 float height= texture2D(offsetMap,
                 vec2(s,t) * config[3].st).x;
 // compute reference plane...
 float plane= dot(point.xyz, N)-
              dot(N, O)+(height*config[3].z );
 // perform clipping if surface is enabled...
 return (plane < 0.0 && bool(config[3].w));
}
```

Listing 1: GLSL implementation to evaluate 2.5D clip-surface for a given point.

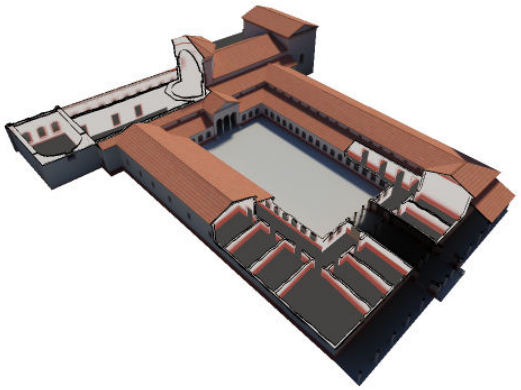


Figure 6: Rendering multiple cut-away views of a virtual 3D building model by applying two 2.5D clip-surfaces.

4.5 Multiple Clip-Surfaces

To enable multiple cut-away views, a number of clip-surfaces are evaluated within a single rendering pass. Figure 6 shows an example for applying two different clip-surfaces to reveal parts of the building footprint and internal structures, such as walls and doorways, which otherwise would be hidden to the viewer.

The quality of the 3D models (in terms of modeled interior, solid walls, and consistency of polygon orientation), is important for the resulting visual impression. Besides additional configuration issues, the application of cut-away views introduces a number of challenges and technical implications to the visualization framework: for example, Figure 6 shows shading and shadow discontinuities for areas inside and outside the building. These are caused by using a pre-computed lighting approach, which is only valid for views from outside the building. This effect can be compensated partially by using image-based global lighting approaches that approximate ambient occlusion [21].

5 CAP SURFACES

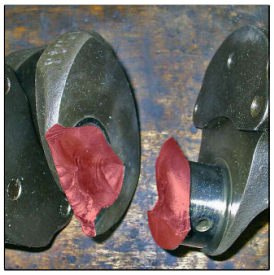


Figure 7: Surfaces of a broken real-world crank marked in red.

For certain types of renderings, it can be desired to show the inner material properties after applying clipping. The sections marked in red in Figure 7 represent the surface of the inner material of a broken crank denoted as *caps*. For planar clipping planes, capping is usually performed by a stencil-buffer technique using the back-face and front-face polygon orientation information [3]: a stencil mask representing the visible back-faces determine the areas where to render the cap. Since the non-regularity of the clip surface, this approach is not suitable to be used for our purposes.

This paper presents an image-based approach for the real-time rendering of cap surface for a 2.5D clip-surface. It is suitable for any solid input mesh, i.e., a mesh that is modeled "water-tight" in real-time.

5.1 Rendering of Cap Surfaces

Conceptually, the rendering of cap surfaces comprises two main steps: the *generation* of a cap surface and the *clipping* of all surface parts that are *outside* the volume enclosed by the input mesh.

During the *cap-surface generation step*, a triangulated quad is generated using the U and V parameters, which resembles the 2.5D cap surface (cf. to *Derived Cap-Surface* in Fig. 3). This mesh is then adaptively refined on GPU [4] to achieve a sufficient vertex density. Here, a generic refinement pattern (stored at full resolution as a vertex buffer on the GPU memory) is used to virtually create additional inner vertices for the generated cap-surface. Subsequently, each vertex of the refined mesh is displaced using the offset map *OM* as well as textured and shaded using specific diffuse, normal, and occlusion maps. The generation of texture coordinates is similar as for the clipping approach (Eqn. 1 to 3).

For the *cap-surface clipping step* during the rasterization of the refined cap surface, each fragment is tested if it lies inside or outside the volume enclosed by the input mesh in order to determine where the gaps are located that have to be covered by the surface. Fragments that fail a *volumetric parity test* (Section 5.3), which is based on an *image-based volumetric representation* of the input mesh (Section 5.2), are discarded using a fragment shader program (cf. to *Clipped Cap-Surface* in Fig. 3 and Section 5.4 for implementation).

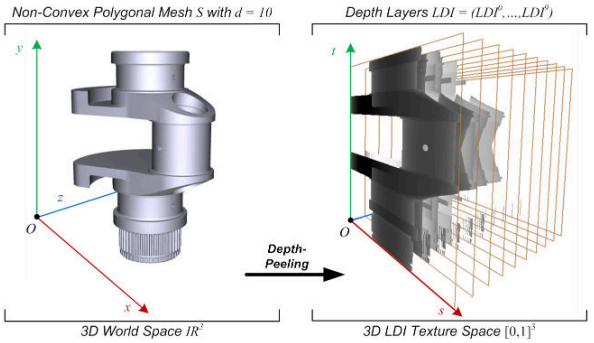


Figure 8: Example of an volumetric depth sprite. The non-convex polygonal mesh S is depth-peeled into layers of unique depth complexity.

5.2 Volumetric Depth Sprites

A *volumetric depth sprite* (VDS) is an image-based representation of a shapes volume by storing its depth values along parallel viewing rays [30]. A VDS extends the concept of LDIs [29] that contain layers of unique depth complexity. An LDI is a view of the scene from a single input camera view, but with multiple pixels

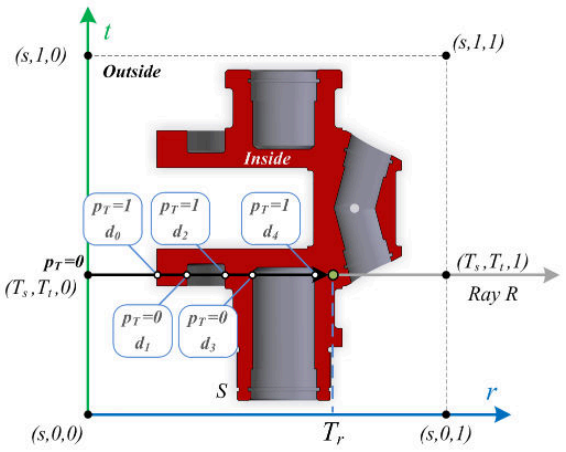


Figure 9: Ray marching through an LDI representation of the complex input shape shown in Figure 8. A ray R intersects the depth layers LDI^i at five points and adjusts the rays parity p_T accordingly.

along each line of sight. The size of this representation grows linearly with the observed depth complexity in the scene. Figure 8 shows an example of a VDS derived from a complex 3D shape.

A VDS representation consists of the following components: $VDS = (G, LDI, d, w_i, h_i)$. Here, $G \in \mathbb{R}^3$ denotes the reference position of the VDS in world-space coordinates. The depth complexity of S is denoted as $d \in \mathbb{N}_{/\{0,1\}}$. The layered depth image consist of d depth maps $LDI = (LDI^0, \ldots, LDI^{d-1})$. The initial texture resolution of width and height is given by $w_i, h_i \in \mathbb{N}$. To obtain a depth value $d_i \in [0,1] \subset \mathbb{R}, 0 \leq i \leq d-1$ in the i^{th}-depth layer for a 2D point $(s,t) \in [0,w_i] \times [0,h_i]$, the 3D texture is sampled in LDI texture space using the coordinates $LDI^i_{(s,t)} = (s,t,i)$.

The creation of a VDS is performed within a preprocessing step using multi-pass render-to-texture in combination with depth-peeling [8]. Given a solid polygonal mesh S, the associated LDI can be generated by performing the following three steps:

1. The shape is scaled uniformly to fit into the unit volume $[0,1]^3$. A camera orientation and an orthogonal projection is chosen that covers this unit volume with the near and far clipping planes adjusted accordingly.

2. The depth complexity d is computed and a 3D texture or 2D texture array is created with an initial resolution of width w_i, height h_i, and depth d.

3. Finally, depth-peeling [8] in combination with RTT is performed. The solid S is peeled using linearized depth values using a W-buffer [17].

5.3 Volumetric Parity Test

Real-time volumetric tests enable a binary partition of a given arbitrary shape on vertex, primitive, and fragment level [30]. They have a number of applications in real-time rendering and interactive visualization, such as pixel-precise clipping, collision detection, and rendering with hybrid styles [12]. Given a VDS, a *volumetric parity test* (VPT) classifies a point P with respect to its position in the shape's volume: it can either be *inside* or *outside*. To model such test, a Boolean *coordinate parity* $p_T \in \{0,1\}$ is used. Before testing P, it must be transformed into the specific 3D LDI texture space. For a point P in world-space coordinates, the transformed coordinate $T = (T_s, T_t, T_r)$ can be obtained by $T = \mathbf{M} \cdot P$.

The matrix $\mathbf{M}$ is defined by $\mathbf{M} := \mathbf{T}(C) \cdot \mathbf{S} \cdot \mathbf{B} \cdot \mathbf{T}(-G)$. Where $\mathbf{B}$ is a rotated ortho-normal basis of the VDS. P is transformed into the LDI texture coordinate space ($\mathbf{B} \cdot \mathbf{T}(-G)$), scaled by $\mathbf{S}$, and translated ($\mathbf{T}(C)$) into the LDI origin $C = (0.5, 0.5, 0.5)$.

Subsequently, a ray $R = [(T_s, T_t, 0)(T_s, T_t, 1)]$ is constructed that marches through the depth layers LDI^i and compares T_r with the stored depth values. Starting with an initial parity, p_T is swapped every time R crosses a layer of unique depth complexity (cf. Figure 9). This test can be formulated as $p_T = VPT(T, LDI)$ with:

$$VPT(T, LDI) = \begin{cases} 1, & \exists d_i \wedge \exists d_{i+1} : d_i \leq T_r < d_{i+1} \\ 0, & otherwise \end{cases}$$

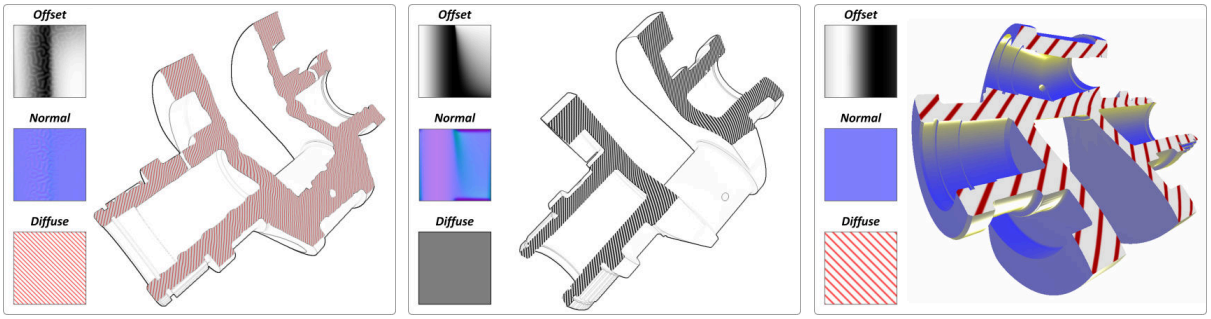
$$d_i \in LDI^i_{(T_s,T_t)} \qquad d_{i+1} \in LDI^{i+1}_{(T_s,T_t)}$$

5.4 Fragment Shader Implementation

The implementation of the ray-marching algorithm needs to iterate over the number of depth maps stored in the 3D texture, which represents the LDI. Listing 2 shows the GLSL source code that implements the VPT. The performance of this algorithm depends on the number of samples the VPT has to perform. The presented volumetric test consists of less than 20 assembler instructions per executed loop.

```
bool volumetricParityTest(
 const in vec4 T, // coordinate in LDI-space
 const in sampler3D LDI, // layered depth image
 const in int depth, // depth complexity ds
 const in bool initParity) // initial parity p
{ // initial parity; true = outside
  bool parity = initParity;
  // compute offset to address texture slices
  float offset = 1.0 / float(depth);
  // for each texture layer do
  for(float i = 0.0; i < float(depth); i++){
    if(T.r < texture3D(LDI, // perform depth test
        vec3(T.st, offset * i)).x) {
      parity = !parity; // swap parity
    }//endif }//endfor
  return parity;
}
```

Listing 2: GLSL implementation of the VPT.



(a) (b) (c)

Figure 10: Examples for applying different parameterizations of a 2.5D clip-surface to a virtual 3D model of a crank. The insets show the respective 2D offset, normal, and diffuse texture maps used for stylization.

6 RESULTS & DISCUSSION

6.1 Application Examples

Figure 10 shows results for the proposed rendering techniques using different offset, normal, diffuse, and offset texture maps that enable stylization and appearance variations. For example, technical illustration styles can be achieved using hatch textures for the cap surfaces in combination with image-based edge enhancement [22] (cf. Figure 10(a) and 10(b)) or Gooch Shading [10] (cf. Figure 10(c)). Further, Figure 11 shows an application example that uses solid 3D textures [16] in combination with occlusion maps for texturing the cap surface.

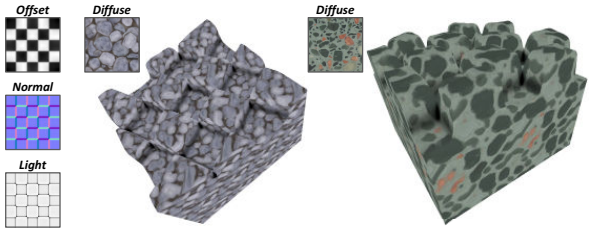


Figure 11: A single 2.5D clip-surface is applied to a cube using light maps and solid textures for the stylization of the cap-surface.

6.2 Rendering Performance

We tested the performance of a clip-surface with a single cap-surface using a NVIDIA GeForce GTX 285 GPU with 2048 MB video RAM on a Intel Xeon CPU with 2.33 GHz and 3 GB of main memory a viewport resolution of 1024×768 pixels. The test model (Figure 10) comprises 50,394 vertices and 99,999 faces. It performs with 131 frames-per-second (FPS) without clipping in comparison to 125 FPS with clipping and cap-surface enabled (with a sub-division level of 256 and an LDI resolution of 1024^2 pixels). The applied cap-surface technique is fill-limited, i.e., the runtime performance depends on the number of fragments tested against the volumetric depth sprite.

6.3 Limitations & Improvements

The presented approach implies a number of technical and conceptual limitations. Despite requiring a watertight mesh, the rendering technique is limited by two drawbacks: (1) it requires an additional data structure (VDS), which is created during a preprocessing step and is only suitable for static meshes; and (2) to obtain a high visual quality, a sufficient vertex density of the cap surface is required. With respect to this, matching the tessellation factor to the screen resolution, which is required to avoid gaps between the clipped mesh and the cap surface, is an open research question.

With respect to the current implementation, there are numerous ways for future work. Instead of using volumetric depth sprites, which are generated in a preprocessing step, one could perform the volumetric parity test based on a k-Buffer implementation [1] that is generated per-frame and per-object. Additionally, this enables also the usage of dynamic objects. Furthermore, to achieve a sufficient vertex density for the mesh that represents the cap surfaces, hardware-accelerated and programmable tessellation [24] could be of interest. With respect to this, the application of relief mapping [25] to render micro structures of the cut surface could be researched.

7 CONCLUSIONS

This paper presents a concept and real-time rendering technique for implementing interactive 2.5D clip-surfaces. The implementation is fully hardware accelerated and includes an approach for rendering cap surfaces that are applicable to arbitrary solid input meshes. The main drawbacks of our approach are the necessary additional data structure, that consumes additional GPU memory as well as the need for highly tessellated surfaces in order to avoid rendering artifacts for the cap surfaces. However, these drawbacks can be addressed and counter-balanced in future implementations using advanced GPU capabilities.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the paper. This work was funded by the German Federal Ministry of Education and Research (BMBF), as part of the InnoProfile Transfer research group "4DnD-Vis".

REFERENCES

[1] Louis Bavoil, Steven P. Callahan, Aaron Lefohn, João L. D. Comba, and Cláudio T. Silva. Multi-fragment effects on the gpu using the k-buffer. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 97–104, New York, NY, USA, 2007. ACM.

[2] Aasmund Birkeland, Stefan Bruckner, Andrea Brambilla, and Ivan Viola. Illustrative membrane clipping. *Computer Graphics Forum*, 31(3):905–914, June 2012.

[3] David Blythe, Tom McReynold, Brad Grantham, Mark J. Kilgard, and Scott R. Nelson. Programming with OpenGL: Advanced Rendering. In A. Rockwood, editor, *SIGGRAPH Course*, New York, NY, USA, 1999. ACM Press.

[4] Tamy Boubekeur and Christophe Schlick. Generic Mesh Refinement on GPU. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '05, pages 99–104, New York, NY, USA, 2005. ACM Press.

[5] Michael Burns and Adam Finkelstein. Adaptive cutaways for comprehensible rendering of polygonal scenes. *ACM Trans. Graph.*, 27(5):154:1–154:7, December 2008.

[6] Michael Clifton and Alex Pang. Cutting planes and beyond. *Comput. Graph.*, 21(5):563–575, September 1997.

[7] Chris Coffin and Tobias Hollerer. Interactive perspective cutaway views for general 3d scenes. In *Proceedings of the 3D User Interfaces*, 3DUI '06, pages 25–28, Washington, DC, USA, 2006. IEEE Computer Society.

[8] Cass Everitt. Interactive Order-Independent Transparency. Technical report, NVIDIA Corporation, June 2001.

[9] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[10] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 447–452, New York, NY, USA, 1998. ACM.

[11] Günther Greiner and Kai Hormann. Efficient clipping of arbitrary polygons. *ACM Trans. Graph.*, 17(2):71–83, April 1998.

[12] Roland Jesse and Tobias Isenberg. Use of Hybrid Rendering Styles for Presentation. In *Poster Proceedings of WSCG 2003*, pages 57–60, 2003. Short Paper.

[13] Olga Karpenko, Wilmot Li, Niloy Mitra, and Maneesh Agrawala. Exploded view diagrams of mathematical surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1311–1318, November 2010.

[14] Florian Kirsch and Jürgen Döllner. Opencsg: A library for image-based csg rendering. In *USENIX 2005 Annual Technical Conference, FREENIX Track*, pages 129–140, 2005.

[15] Sebastian Knödel, Martin Hachet, and Pascal Guitton. Interactive Generation and Modification of Cutaway Illustrations for Polygonal Models. 2009.

[16] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):2:1–2:9, 2007.

[17] Eugene Lapidous and Guofang Jiao. Optimal Depth Buffer for Low-Cost Graphics Hardware. In *HWWS '99*, pages 67–73, New York, NY, USA, 1999. ACM Press.

[18] Wilmot Li, Maneesh Agrawala, Brian Curless, and David Salesin. Automated generation of interactive 3d exploded view diagrams. *ACM Trans. Graph.*, 27(3):101:1–101:7, August 2008.

[19] Wilmot Li, Lincoln Ritter, Maneesh Agrawala, Brian Curless, and David Salesin. Interactive Cutaway Illustrations of Complex 3D Models. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM Press.

[20] William E. Lorensen. Geometric clipping using boolean textures. In *Proceedings of the 4th conference on Visualization '93*, VIS '93, pages 268–274, Washington, DC, USA, 1993. IEEE Computer Society.

[21] Thomas Luft, Carsten Colditz, and Oliver Deussen. Image Enhancement by Unsharp Masking the Depth Buffer. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 1206–1213, New York, NY, USA, 2006. ACM Press.

[22] Marc Nienhaus and Jürgen Döllner. Edge-enhancement - an algorithm for real-time non-photorealistic rendering. *International Winter School of Computer Graphics, Journal of WSCG*, 11(2):346–353, 2003.

[23] Marc Nienhaus, Florian Kirsch, and Jürgen Döllner. Illustrating design and spatial assembly of interactive csg. In *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, AFRIGRAPH '06, pages 91–98, New York, NY, USA, 2006. ACM.

[24] Matthias Nießner, Charles T. Loop, Mark Meyer, and Tony DeRose. Feature-adaptive gpu rendering of catmull-clark subdivision surfaces. *ACM Trans. Graph.*, 31(1):6, 2012.

[25] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 155–162, New York, NY, USA, 2005. ACM.

[26] Wen Qi and Jean-Bernard Martens. Tangible user interfaces for 3d clipping plane interaction with volumetric data: a case study. In *Proceedings of the 7th international conference on Multimodal interfaces*, ICMI '05, pages 252–258, New York, NY, USA, 2005. ACM.

[27] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the symposium on Data visualisation 2003*, VISSYM '03, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[28] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.*, 24(4):197–206, September 1990.

[29] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered Depth Images. In *SIGGRAPH '98*, pages 231–242, New York, NY, USA, 1998. ACM Press.

[30] Matthias Trapp and Jürgen Döllner. Real-time volumetric tests using layered depth images. In K. Mania and E. Reinhard, editors, *Eurographics 2008 Shortpaper*, pages 235–238. The Eurographics Association, 2008.

[31] Daniel Weiskopf, Klaus Engel, and Thomas Ertl. Volume clipping via per-fragment operations in texture-based volume visualization. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 93–100, Washington, DC, USA, 2002. IEEE Computer Society.

[32] Daniel Weiskopf, Klaus Engel, and Thomas Ertl. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):298–312, July 2003.